INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	*Fundamentos de la programación 02*				
TÍTULO DE LA PRÁCTICA:	*Arreglos bidimensionales a objetos*				
NÚMERO DE PRÁCTICA:	*05*	**AÑO LECTIVO:**	*2024-B*	**NRO. SEMESTRE:**	*II*
FECHA DE PRESENTACIÓN	*18/10/2024*	**HORA DE PRESENTACIÓN**	*18:00:00*		
INTEGRANTE (s) *Mauro Snayder Sullca Mamani*				**NOTA (0-20)**	
DOCENTE(s): *Ing. Lino Jose Pinto Oppe*					

RESULTADOS Y PRUEBAS

I. EJERCICIOS RESUELTOS:

```java
/**
 *
 * @author Mauro Snayder
 */
public class Soldado {
    // Creamos los atributos
    private String nombre="________";
    private int vida;
    private int fila;
    private int columna;

    // Creamos los Set y Get de cada atributo
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getVida() {
        return vida;
    }

    public int getFila(){
        return fila;
    }

    public int getColumna(){
        return columna;
    }
```

```java
    //Generamos una posicion aleatoria para el soldado
    public void aleatorioPosicion(int fila,int columna){
        this.fila=(int)(Math.random()*fila);
        this.columna=(int)(Math.random()*columna);
    }
    //Generamos la vida del soldado
    public void aleatorioVida(){
        this.vida=(int)(Math.random()*5+1);
    }
    // Creamos el toString
    public String toString() {
        return "Soldado{" + "nombre=" + nombre + ", vida=" + vida + ", fila=" + fila + ", columna=" + columna + "}\n";
    }
}
```

```java
/**
 *
 * @author Usuario24B
 */

public class Actividad05 {
    public static void main(String[] args) {
        // Inicializamos dos ejércitos con un número aleatorio de soldados entre 1 y 5
        int fila=(int)(Math.random()*5+1);//Generamos la fila de la tabla
        int columna=(int)(Math.random()*5+1);//Generamos la columna de la tabla
        int numSoldadosPrevio=(int)(Math.random()*10+1);//Generamos la cantidad de soldados
        //Tomamos el valor menor entre la cantidad de soldados y el numero de espacios que tiene la tabla
        //ya que puede ocurrir que la cantidad de soldados sea mayor a los espacios de la tabla, lo cual
        //daria un error
        int numSoldados=Math.min(numSoldadosPrevio,fila*columna);
        Soldado[][] ejercitol=new Soldado[fila][columna];
        for (int i=0;i<ejercitol.length;i++){//inicializamosel arreglo con soldados sin datos
            for (int j=0;j<ejercitol[i].length;j++)
                ejercitol[i][j]=new Soldado();
        }
        inicializarEjercito(ejercitol,numSoldados);
        mostrarEjercitoTabla(ejercitol);
        System.out.print("\nEl soldado con mayor vida es: "+mayorVida(ejercitol).toString());
        System.out.println("El promedio de la vida del ejercito es: "+vidaTotal(ejercitol)/numSoldados);
        System.out.println("La vida total del ejercito es: "+vidaTotal(ejercitol));
        System.out.println("\nLista de los soldados por orden de creacion: ");
        mostrarEjercitoOrdenCreacion(ejercitol,numSoldados);
        System.out.println("\nEl ranking de los soldados es: ");
        rankingSoldados(ejercitol,numSoldados);

    }
    // Método para inicializar una tabla con cierto numeros de soldados
    public static void inicializarEjercito(Soldado[][] ejercito,int numSoldados){
        for (int i=0;i<numSoldados;i++){
            Soldado persona=new Soldado();//creamos "persona" para luego ponerlo dentro del arreglo
            persona.setNombre("soldado"+i);
            do {
                persona.aleatorioPosicion(ejercito.length,ejercito[0].length);//generamos una posicion
                persona.aleatorioVida();//generamos la vida
            }
            //El bucle se repite si en una posicion aleatoria ya existe un soldado puesto
            while(!ejercito[persona.getFila()][persona.getColumna()].getNombre().equals("_______"));
            ejercito[persona.getFila()][persona.getColumna()]=persona;//ponemos el soldado dentro del arreglo
        }
    }
```

```java
    // Metodo para mostrar la tabla
    public static void mostrarEjercitoTabla(Soldado[][] ejercito){
        for (int i=0;i<ejercito.length;i++){
            System.out.print("|");
            for (int j=0;j<ejercito[i].length;j++)
                System.out.print(ejercito[i][j].getNombre()+"|");
            System.out.println();
        }
    }
    //Metodo para determinar el soldado con mayor vida
    public static Soldado mayorVida(Soldado[][] ejercito){
        Soldado mayor=new Soldado();//creamos un objeto para almacenar al soldado mayor
        for (int i=0;i<ejercito.length;i++){
            for (int j=0;j<ejercito[i].length;j++){
                if (ejercito[i][j].getVida()>mayor.getVida())
                    mayor=ejercito[i][j];//actualizamos "mayor" si otro soldado tiene mayor vida
            }
        }
        return mayor;
    }
    //Metodo para determinar la vida total de todos los soldados
    public static double vidaTotal(Soldado[][] ejercito){
        double vidaT=0;//vida inicial
        for (int i=0;i<ejercito.length;i++){
            for (int j=0;j<ejercito[i].length;j++)
                vidaT+=ejercito[i][j].getVida();//sumamos la vida de cada soldado
        }
        return vidaT;
    }
    //Metodo para ver las lista de los soldados por el orden de creacion
    public static void mostrarEjercitoOrdenCreacion(Soldado[][] ejercito,int numSoldados){
        Soldado[] list=new Soldado[numSoldados];//creamos un arreglo unidimensional.
        int copiar=0;
        for (int j=0;j<ejercito.length;j++){//Copiamos el arreglo bidimensional a un unidimensional
            for (int k=0;k<ejercito[j].length;k++){//ya que es mas facil ordenarlos
                if (!ejercito[j][k].getNombre().equals("________")){
                    list[copiar]=ejercito[j][k];
                    copiar++;
                }
            }
        }
        ordenamientoBurbuja(list);//ordenamos el arreglo unidimensional
        for (int i=0;i<list.length;i++){//imprimimos el arreglo
            System.out.print(list[i].toString());
        }
    }
    //Metodo para ver el ranking de los soldados (por vida)
    public static void rankingSoldados(Soldado[][] ejercito,int numSoldados){
        Soldado[] listRankT=new Soldado[numSoldados];//creamos un arreglo unidimensional.
        int copiar=0;
        for (int j=0;j<ejercito.length;j++){//Copiamos el arreglo bidimensional a un unidimensional
            for (int k=0;k<ejercito[j].length;k++){//ya que es mas facil ordenarlos
                if (!ejercito[j][k].getNombre().equals("________")){
                    listRankT[copiar]=ejercito[j][k];
                    copiar++;
                }
            }
        }
        ordenamientoInsercion(listRankT);//ordenamos el arreglo unidimensional
        for (int i=0,j=listRankT.length-1;j>=0;i++,j--){//imprimimos el arreglo
            System.out.print((i+1)+" -> "+listRankT[j].toString());
        }
    }
```

```java
    //Metodo de ordenamiento burbuja para los nombres de los soldados, ya que de esa
    //manera podemos ver el orden de creacion de los soldados
    public static void ordenamientoBurbuja(Soldado[] lista){
        Soldado cambio;
        for(int i=0;i<lista.length-1;i++){
            for(int j=0;j<lista.length-i-1;j++){
                if(lista[j].getNombre().compareTo(lista[j+1].getNombre())>0){
                    cambio=lista[j];
                    lista[j]=lista[j+1];
                    lista[j+1]=cambio;
                }
            }
        }
    }
    //Metodo de ordenamiento de insersion para la vida de los soldados
    public static void ordenamientoInsercion(Soldado[] lista) {
        for (int i=1;i<lista.length;i++) {
            Soldado soldadoActual=lista[i];
            int j=i-1;
            while (j>=0 && lista[j].getVida()>soldadoActual.getVida()){
                lista[j+1]=lista[j];
                j--;
            }
            lista[j+1]=soldadoActual;
        }
    }
}
```

II. PRUEBAS

¿Con que valores comprobaste que tu práctica estuviera correcta?

Se utilizaron una tabla 10x10 , así como el número de soldados, variando la posición entre 1 y 5 para las filas y columnas y entre 1 y 10 para la cantidad de soldados. Esto permitió evaluar cómo el programa manejaba diferentes configuraciones y asegurarse de que los soldados se asignaran correctamente a las posiciones en la tabla.

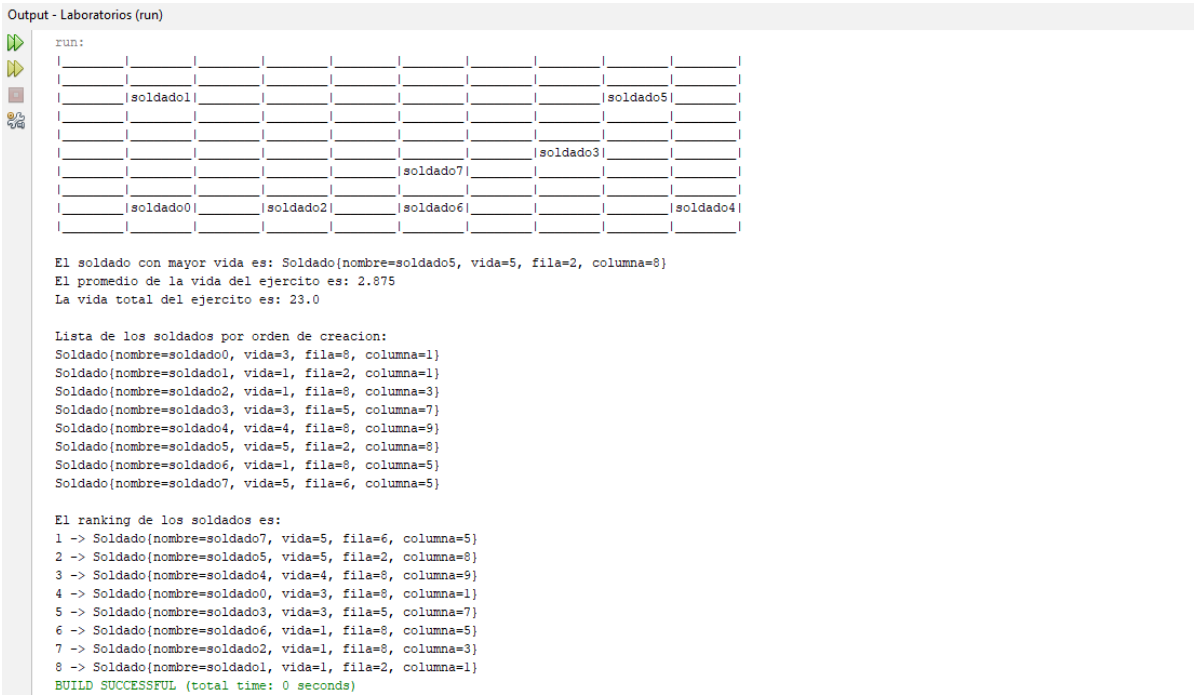
¿Qué resultado esperabas obtener para cada valor de entrada?

Se esperaba que, dependiendo de los valores de entrada, la tabla se llenara correctamente sin solapamientos de soldados. Además, se anticipaba que el programa identificara el soldado con mayor vida, calculase la vida total y el promedio de vida, y que mostrara la lista de soldados en el orden de creación, así como su ranking por vida.

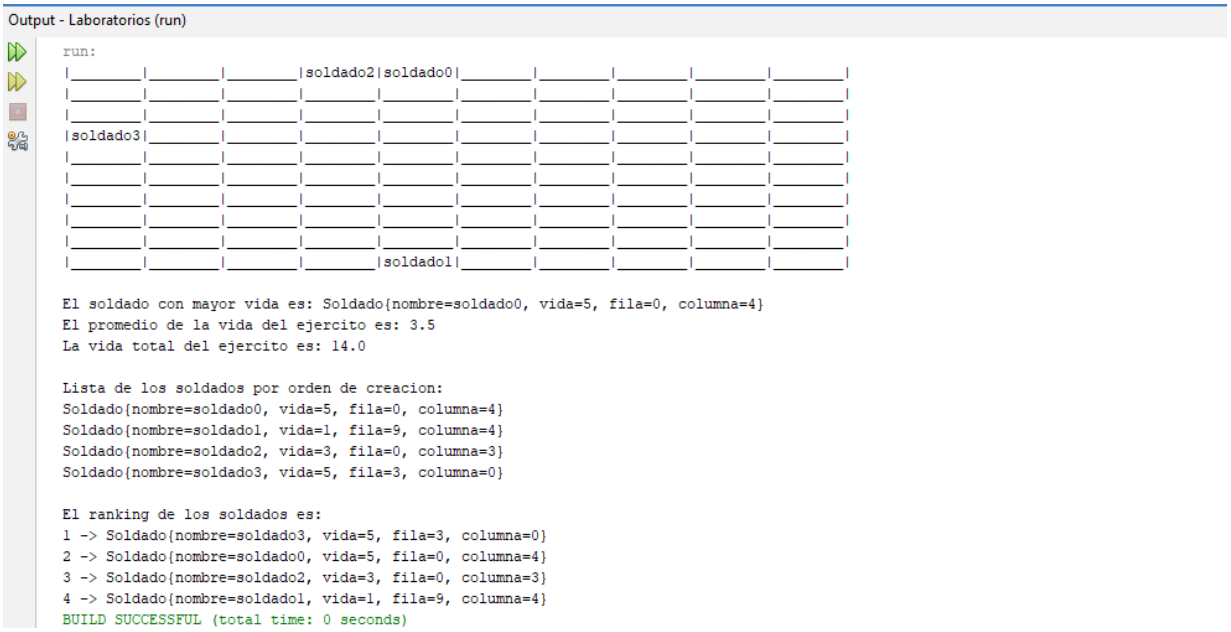
¿Qué valor o comportamiento obtuviste para cada valor de entrada?

Al ejecutar el programa, se observó que los resultados variaban por la aleatoriedad en las posiciones y vidas de los soldados. La lista de soldados se presentó en el orden de creación, y el ranking se mostró correctamente, indicando que el programa funcionó de acuerdo con lo esperado en general.

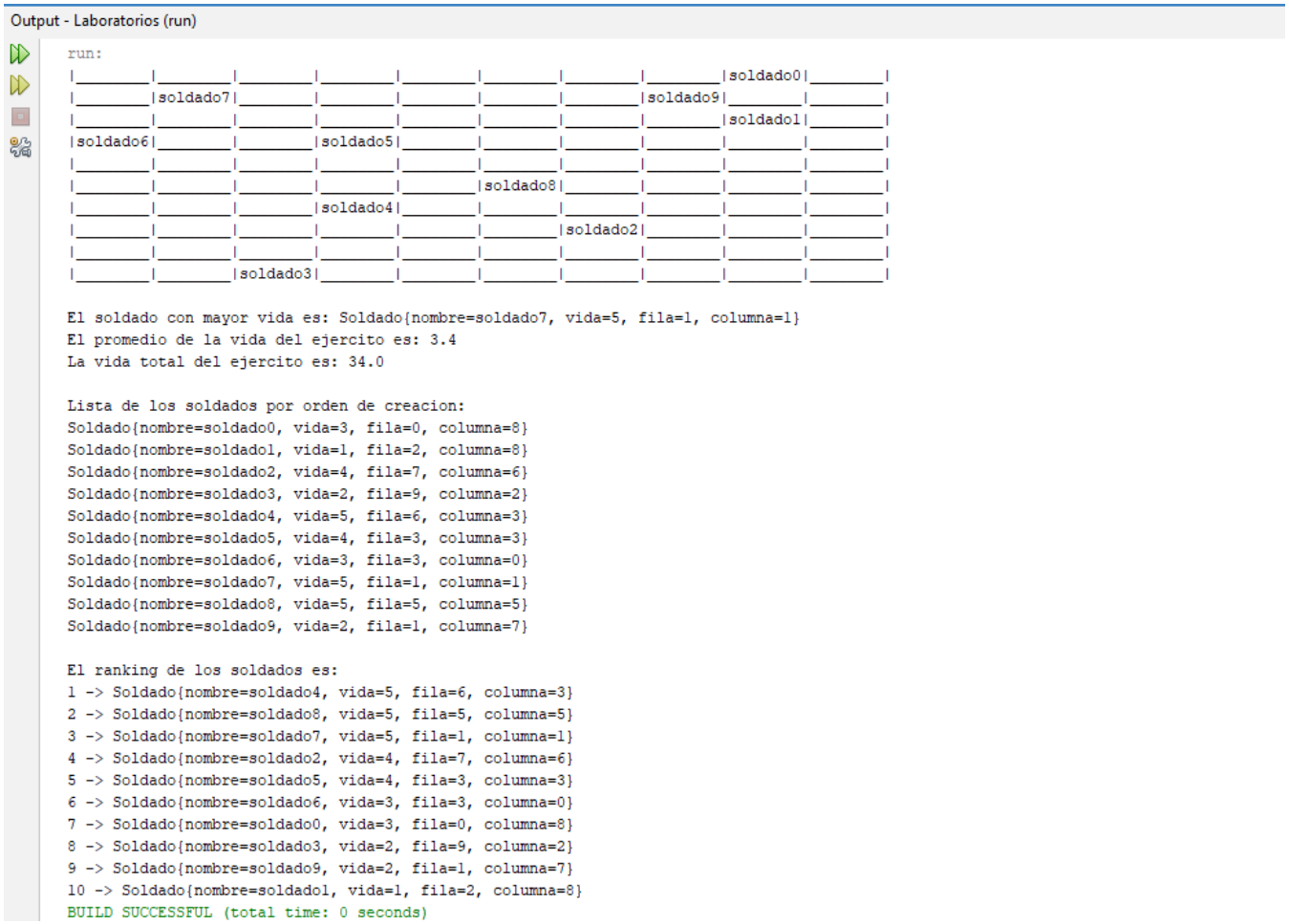
La primera ejecución del programa podemos ver que se generó una tabla 10x10 y 8 soldados. El soldado con mayor vida es el "soldado5", el promedio de vida es "2.875" y la vida total es "23". Y por último nos muestra la lista de los soldados por orden de creación y por orden de vida.



La segunda ejecución del programa podemos ver que se generó una tabla 10x10 y 4 soldados. El soldado con mayor vida es el "soldado0", el promedio de vida es "3.5" y la vida total es "14". Y por último nos muestra la lista de los soldados por orden de creación y por orden de vida.

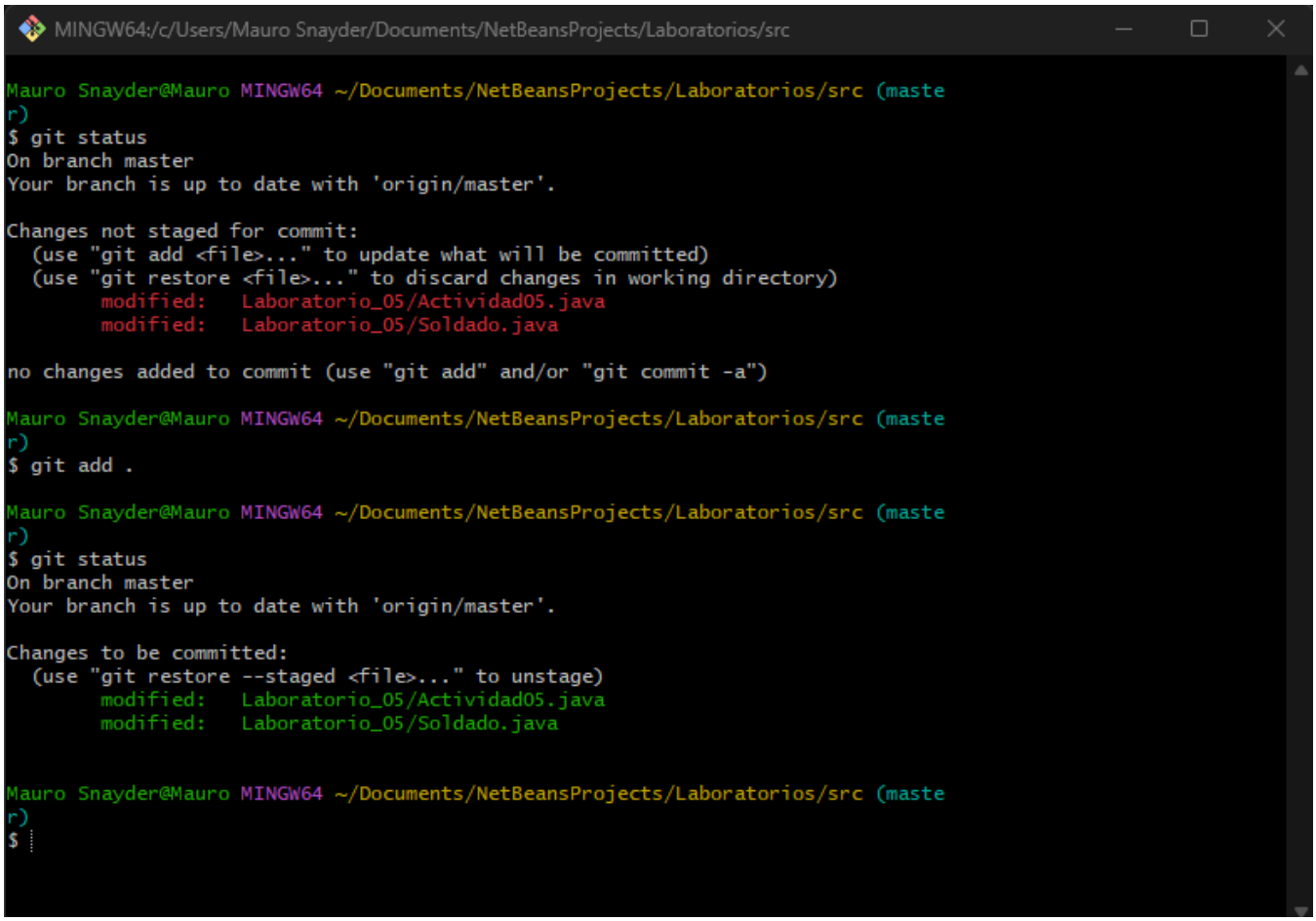


La tercera ejecución del programa podemos ver que se generó una tabla 10x10 y 10 soldados. El soldado con mayor vida es el “soldado7”, el promedio de vida es “3.4” y la vida total es “34”. Y por último nos muestra la lista de los soldados por orden de creación y por orden de vida.

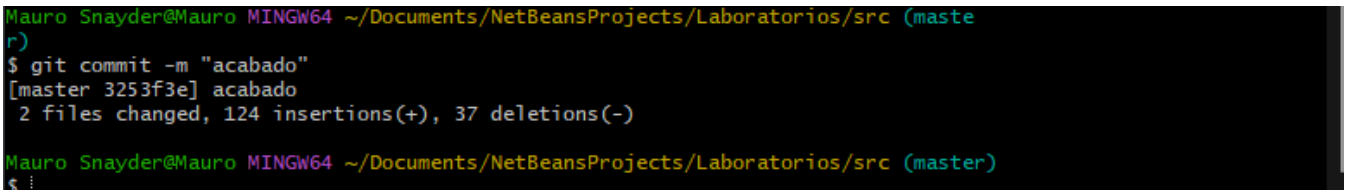


III. COMMITS:

Entramos a nuestra carpeta donde están nuestros archivos y añadimos los cambios.



Hacemos un commit



Subimos nuestro commit a nuestro repositorio remoto

```
Mauro Snayder@Mauro MINGW64 ~/Documents/NetBeansProjects/Laboratorios/src (master)
$ git push -u origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 20 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 2.26 KiB | 2.26 MiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/MauroSullcaMamani/FDLP2_LAB.git
   9305b36..3253f3e  master -> master
branch 'master' set up to track 'origin/master'.

Mauro Snayder@Mauro MINGW64 ~/Documents/NetBeansProjects/Laboratorios/src (master)
$
```

Link de mi repositorio: https://github.com/MauroSullcaMamani/FDLP2_LAB.git

IV. RUBRICA:

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	✓	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	✓	3	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	✓	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	✓	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	✓	2	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	✓	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	✓	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	✓	3	
TOTAL		20		18	

Tabla 2: Rúbrica para contenido del Informe y demostración

CONCLUSIONES

En conclusión, la utilización de arreglos bidimensionales de objetos en el programa, junto con técnicas de inicialización, búsqueda y ordenamiento, permite manejar colecciones de soldados de manera eficiente y organizada. Al trabajar con objetos, se pueden gestionar múltiples atributos, como nombre y vida, de forma estructurada, lo que facilita la manipulación y el acceso a la información. Las funcionalidades implementadas para identificar al soldado con mayor vida, calcular la vida total y promedio, y ordenar soldados por su creación y vida, mejoran la accesibilidad y optimizan el rendimiento del programa, proporcionando una base sólida para futuras mejoras.

METODOLOGÍA DE TRABAJO

Lo primero que hice, es leer cada los enunciados de cada ejercicio y también tomar en cuenta las restricciones que nos da para así poder buscar una solución al problema. Después observar el código y entender la funcionalidad de cada uno y completar las partes que están incompletas. Y por último comprobar nuestro código ingresando varias veces valores de prueba para ver que nuestro código está funcionando correctamente.

REFERENCIAS Y BIBLIOGRAFÍA

E. G. Castro Gutiérrez and M. W. Aedo López, Fundamentos de programación 2: tópicos de programación orientada a objetos, 1st ed. Arequipa, Perú: Universidad Nacional de San Agustín, 2021, pp. 170, ISBN 978-612 5035-20-2.